Variational Canonical Correlation Analysis (VCCA)

Yang Chen

March 29, 2018

1 Multi-View Learning task

In the multi-view representation learning setting, we have multiple views (types of measurements) of the same under- lying signal, and the goal is to learn useful features of each view using complementary information contained in both views. The learned features should uncover the common sources of variation in the views, which can be helpful for exploratory analysis or for downstream tasks.

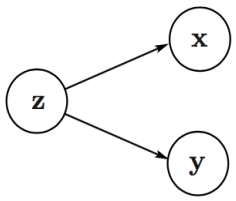
2 VCCA

VCCA models the **joint distribution of two views** with a latent variable model capturing both the shared and private information, where the distributions are parameterized by deep neural networks. VCCA optimizes a **variational lower bound** of the likelihood and allows for straightforward training using small minibatches of training samples.

3 Latent Variable

3.1

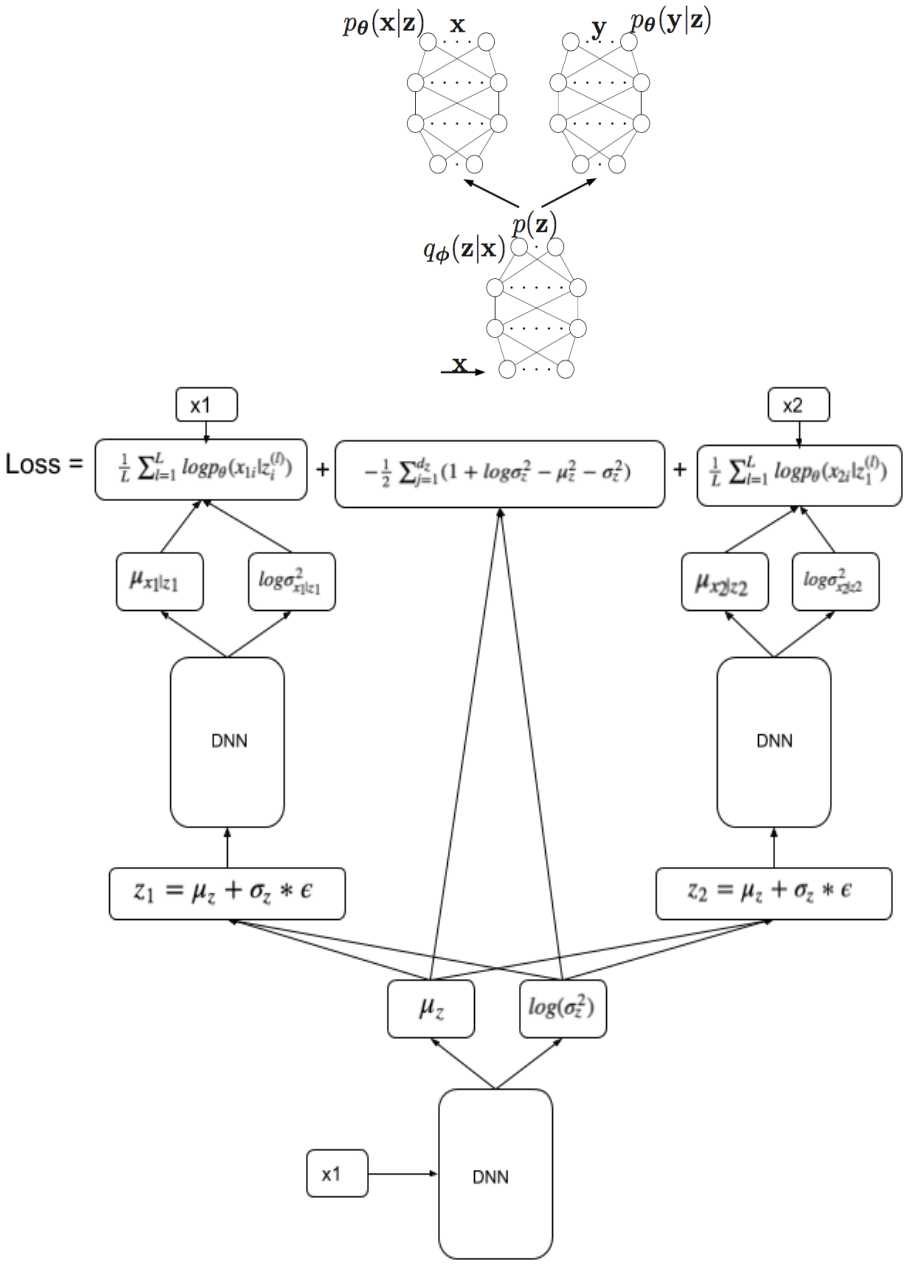
The probabilistic latent variable model interpretation of linear CCA is shown hare:



$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mathbf{W}_x\mathbf{z}, \mathbf{\Phi}_x)$$
$$\mathbf{y}|\mathbf{z} \sim \mathcal{N}(\mathbf{W}_y\mathbf{z}, \mathbf{\Phi}_y)$$

Assuming that x and y are linear functions of some random variable z in R^{d_z}, where $d_z <= min(d_x, d_y)$. The prior distribution p(z) and conditional distributions $p(x|z)$ and $p(y|z)$ are Gaussian. Jordan showed that $E(z|x)$ lives in the same space as the linear CCA projection for x.

The VCCA model extends the latent variable interpretation of linear CCA to nonlinear observation models parameterized by DNN.



The probabilistic latent variable model of CCA defines the following joint distribution over the random variables (x,y):

$p(x, y, z) = p(z)p(x|z)p(y|z)$

$p(x, y) = \int p(x, y, z)dz$

Assumption: x and y are conditionally independent on z.

However, $p_\theta(x, y)$ does not have a closed form, and $p_\theta(z|x)$ is intractable. (Reason: This integral requires exponential time to compute as it needs to be evaluated over all configurations of **latent variables**. So we need to approximate this posterior distribution.)

4 Objective

4.1 Lower Bound

$logp_\theta(x, y) >= L(x, y; \theta, \Phi) := -D_{KL}(q_\Phi(z|x)||p(z)) + Eq_\Phi(z|x)[logp_\theta(x|z) + logp_\theta(y|z)]$

4.2 Derivation of the lower bound

$$
\begin{aligned}
& log_{p\theta}(x,y) \\
&= log_{p\theta}(x,y)\int q_\Phi(z|x)dz \\
&= \int log_{p\theta}(x,y)q_\Phi(z|x)dz \\
&= \int q_\Phi(z|x)(log(p(x,y,z)/p(z|x,y)))dz \\
&= \int q_\Phi(z|x)(log(p(x,y,z)/q(z|x)*q(z|x)/p(z|x,y))dz \\
&= \int q_\Phi(z|x)log\frac{p(x,y,z)}{q(z|x)}dz + \int q_\Phi(z|x)log\frac{q(z|x)}{p(z|x,y)}dz \\
&= E_{q\Phi(z|x)}[log\frac{p_\theta(x,y,z)}{q_\Phi(z|x)}] + D_{KL}(q_\Phi(z|x)||p_\theta(z|x,y)) \\
&>= E_{q\Phi(z|x)}[log\frac{p_\theta(x,y,z)}{q_\Phi(z|x)}] \rightarrow \text{(KL is nonnegative)} \\
&= L(z,y;\theta,\Phi)
\end{aligned}
$$

So

$$
\begin{aligned}
& L(x,y;\theta,\Phi) \\
&= \int q_\Phi(z|x)[log\frac{p(z)p(x|z)p(y|z)}{q(z|x)}]dz \\
&= \int q_\Phi(z|x)[log\frac{p(z)}{q(z|x)} + logp(z|x)logp(y|z)]dz \\
&= -D_{KL}(q_\Phi(z|x)||p(z)) + E_{q(z|x)}[logp(x|z) + logp(y|z)]
\end{aligned}
$$

4.3 Compute E part

$E_{q\Phi(z_i|x_i)}[logp_\theta(x_i|z_i) + logp_\theta(y_i|z_i)] = \frac{1}{L}\sum_{l=1}^{L} logp_\theta(x_i|z_i^{(l)}) + logp_\theta(y_i|z_i^{(l)})$

4.3.1 Code

```
def _compute_reconstr_loss(self, x_input, x_reconstr_mean, x_reconstr_log_sigma_sq, n_out, losstype, STDVAR):

    if losstype == 0:
        # Cross entropy loss.
        reconstr_loss = - tf.reduce_sum(
            x_input * tf.log(1e-6 + x_reconstr_mean) + (1 - x_input) * tf.log(1e-6 + 1 - x_reconstr_mean), 1)
    elif losstype == 1:
        # Least squares loss, with learned std.
        reconstr_loss = 0.5 * tf.reduce_sum(
            tf.div(tf.square(x_input - x_reconstr_mean), 1e-6 + tf.exp(x_reconstr_log_sigma_sq)),
            1) + 0.5 * tf.reduce_sum(x_reconstr_log_sigma_sq, 1) + 0.5 * math.log(2 * math.pi) * n_out
    elif losstype == 2:
        # Least squares loss, with specified std.
        reconstr_loss = 0.5 * tf.reduce_sum(tf.square((x_input - x_reconstr_mean) / STDVAR), 1) + 0.5 * math.log(
            2 * math.pi * STDVAR * STDVAR) * n_out

    # Average over the minibatch.
    cost = tf.reduce_mean(reconstr_loss)
    return cost
```

4.4 Compute KL divergence

$D_{KL}(q_\Phi(z_i|x_i)||p(z_i)) = -\frac{1}{2}\sum_{j=1}^{d_z}(1 + log\sigma_{ij}^2 - \sigma_{ij}^2 - \mu_{ij}^2)$

4.4.1 Derivation of KL divergence

Assumption random variable x1,x2 is Gaussian distribution. $x1 \to N_1(\mu_1, \sigma_1^2), x2 \to N_2(\mu_2, \sigma_2^2)$.

$N(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$

$$
\begin{aligned}
&\int p_1(x) log \frac{p_1(x)}{p_2(x)} dx \\
&= \int p_1(x)(log p_1(x) - log p_2(x)) dx \\
&= \int p_1(x)(log \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{\frac{(x-\mu_1)^2}{2\sigma_1^2}} - log \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{\frac{(x-\mu_2)^2}{2\sigma_2^2}}) dx \\
&= \int p_1(x)(-1/2 log 2\pi - log\sigma_1 - \frac{(x-\mu_1)^2}{2\sigma_1^2} + 1/2 log 2\pi + log\sigma_2 + \frac{(x-\mu_2)^2}{2\sigma_2^2}) dx \\
&= \int p_1(x)(log \frac{\sigma_1}{\sigma_2} + [\frac{(x-\mu_1)^2}{2\sigma_1^2} - \frac{(x-\mu_2)^2}{2\sigma_2^2}]) dx \\
&= \int (log \frac{\sigma_2}{\sigma_1} p_1(x) dx + \int \frac{(x-\mu_1)^2}{2\sigma_1^2} p_1(x) dx - \int \frac{(x-\mu_2)^2}{2\sigma_2^2} p_1(x) dx \\
&= log \frac{\sigma_1}{\sigma_2} + \frac{1}{2\sigma_2^2} \int (x-\mu_2)^2 p_1(x) dx - \frac{1}{2\sigma_1^2} \int (x-\mu_1)^2 p_1(x) dx \\
&= log \frac{\sigma_1}{\sigma_2} + \frac{1}{2\sigma_2^2} \int (x-\mu_2)^2 p_1(x) dx - \frac{1}{2} \to (\int (x-\mu_1)^2 p_1(x) dx = \sigma_1^2) \\
&= log \frac{\sigma_1}{\sigma_2} + \frac{1}{2\sigma_2^2} \int (x-\mu_1+\mu_1-\mu_2)^2 p_1(x) dx - \frac{1}{2} \\
&= log \frac{\sigma_1}{\sigma_2} + \frac{1}{2\sigma_2^2} [\int (x-\mu_1)^2 p_1(x) dx + \int (\mu_1-\mu_2)^2 p_1(x) dx + 2 \int (x-\mu_1)(\mu_1-\mu_2) p_1(x) dx] - \frac{1}{2} \\
&= log \frac{\sigma_1}{\sigma_2} + \frac{1}{2\sigma_2^2} [\int (x-\mu_1)^2 p_1(x) dx + (\mu_1-\mu_2)^2] - \frac{1}{2} \\
&\to (2 \int (x-\mu_1)(\mu_1-\mu_2) p_1(x) dx = (\mu_1-\mu_2) \int (x-\mu_1) p_1(x) dx \\
&= (\mu_1-\mu_2)[\int x p_1(x) dx - \int \mu_1 p_1(x) dx] = (\mu_1 - mu_2)[\mu_1 - \mu_1] = 0 \\
&= log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{2\sigma_2^2} - \frac{1}{2})
\end{aligned}
$$

Let $\mu_2 = 0, \sigma_2^2 = 1$

$$
\begin{aligned}
&= -log\sigma_1 + \frac{\sigma_1^2 + \mu_1^2}{2} - \frac{1}{2} \\
&= -\frac{1}{2}(1 + 2log\sigma_1 - \sigma_1^2 - \mu_1^2) \\
&= -\frac{1}{2}(1 + log\sigma_1^2 - \sigma_1^2 - \mu_1^2)
\end{aligned}
$$

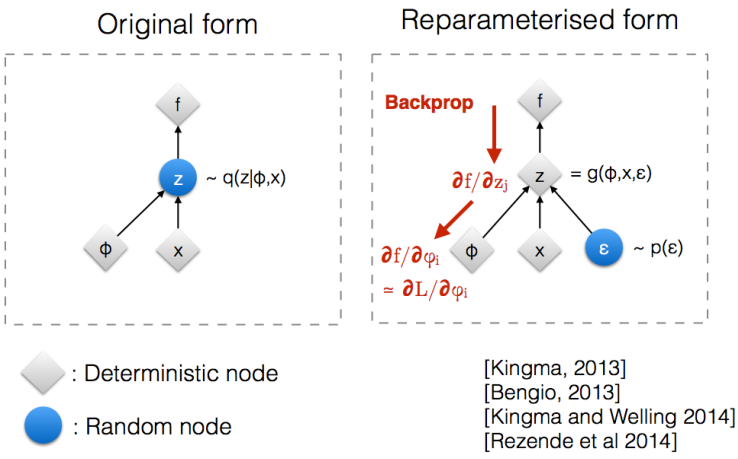
4.4.2 Code

```
latent_loss_z = - 0.5 * tf.reduce_sum(
            1 + self.z_log_sigma_sq - tf.square(self.z_mean) - tf.exp(self.z_log_sigma_sq), 1)
```

5 Reparameterization trick

Intuitively, in its original form, VAEs sample from a random node z which is approximated by the parametric model $q(z|\Phi, x)$ of the true posterior. Backprop cannot flow through a random node.

Introducing a new parameter ϵ allows us to reparameterize z in a way that allows backprop to flow through the deterministic nodes.



5.1 Code

```
# Draw L samples of z.
z_epsshape = tf.mul(tf.shape(self.z_mean), [L, 1])
eps = tf.random_normal(z_epsshape, 0, 1, dtype=tf.float32)
self.z1 = tf.add(tf.tile(self.z_mean, [L, 1]), tf.mul(tf.tile(tf.exp(0.5 * self.z_log_sigma_sq), [L, 1]), eps))
```

6 Deep Neural Network

We consider non-linear observation models $p_\theta(x|z;\theta_x), p_\theta(y|z;\theta_y)$, parameterized by θ_x and θ_y, which can be the collections of weights of DNNs. We also approximate $p_\theta(z|x)$ with the conditional density $q_\Phi(z|x;\Phi_z)$, where Φ_z is the collection of parameters of another DNN.

6.1 Code

```
print("Building view 1 recognition network F ...")
activation = self.x1
width = n_input1
with tf.variable_scope("F", reuse=None, initializer=initializer):
    for i in range(len(architecture["F_hidden_widths"])):
        print("\tLayer %d ..." % (i + 1))
        activation = tf.nn.dropout(activation, self.keepprob)
        if i == (len(architecture["F_hidden_widths"]) - 1):
            weights = tf.get_variable("weights_log_sigma_sq", [width, architecture["F_hidden_widths"][i]])
            biases = tf.get_variable("biases_log_sigma_sq", [architecture["F_hidden_widths"][i]])
            self.z_log_sigma_sq = tf.add(tf.matmul(activation, weights), biases)
        weights = tf.get_variable("weights_layer_" + str(i + 1), [width, architecture["F_hidden_widths"][i]])
        biases = tf.get_variable("biases_layer_" + str(i + 1), [architecture["F_hidden_widths"][i]])
        activation = tf.add(tf.matmul(activation, weights), biases)
        if not architecture["F_hidden_activations"][i] == None:
            activation = architecture["F_hidden_activations"][i](activation)
        width = architecture["F_hidden_widths"][i]
self.z_mean = activation
```

7 Private information

There might be large variations in the input space that can not be explained by the common variables. It may then be beneficial to explicitly model the private variables within each view.

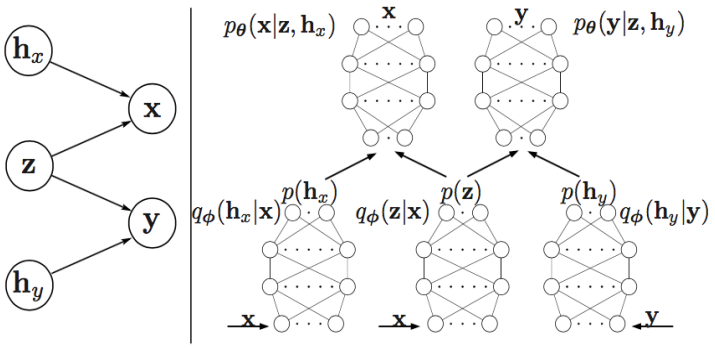


Figure 2. VCCA-private: variational CCA with view-specific private variables.